Modelling and Verification of Multiple UAV Mission Using SMV

Gopinadh Sirigineedi[*], Antonios Tsourdos[†], Rafał Żbikowski[‡], and Brian A. White[§]

Department of Informatics and Sensors,
Cranfield University, Shrivenham,
Swindon SN6 8LA, United Kingdom.

Model checking has been used to verify the correctness of digital circuits, security protocols, communication protocols, as they can be modelled by means of finite state transition model. However, modelling the behaviour of hybrid systems like UAVs in a Kripke model is challenging. This work is aimed at capturing the behaviour of an UAV performing cooperative search mission into a Kripke model, so as to verify it against the temporal properties expressed in Computational Tree Logic (CTL). SMV model checker is used for the purpose of model checking.

1 Introduction

Increase in computational power, improvements in control techniques and other technological advances have led to increased focus on cooperative control of multiple agents in recent years. Cooperating multi-agent systems find application in large number of areas - mobile robots, micro satellite clusters, unmanned aerial vehicles (UAVs), automated highway systems and internet agents. Cooperating multiple-agents offer a large number of benefits such as: increase in the success rate of mission, large area coverage by improvements in latency and information gathering, increase in the computational power offered by distributed computing and graceful degradation in performance

Cooperative UAV control problems that have received recent attention include cooperative formation [18], cooperative task allocation, cooperative path planning, cooperative search and many others. Cooperative search problems have applications in a number of military and civilian applications, such as surveillance and reconnaissance operations, search and rescue; hazard monitoring, battle damage assessment, agricultural coverage tasks and security patrols [26].

Due to the mission critical nature of UAV systems, it is highly important to ensure the correctness of these systems and check if the system meets the design requirements. The failure of control software of the Arian-5 rocket and the Mars rover are hard remainders of what can happen when systems don't perform as per specifications. Verification is the process of verifying the correctness of the system and whether it satisfies the specifications. The common verification processes are *simulation, testing, deductive verification*, and *model checking* [15]. Simulation is performed on the abstract model of the system, where as testing is done on the actual system. Simulation and testing involve giving certain inputs and checking whether the outputs are as expected. These methods are cost effective way to find the errors. However, checking all of the possible interactions and possible faults is almost impossible

[*]PhD Student., DoIS, Cranfield University.
[†]Professor and Head of the Autonomous Systems Group., DoIS, Cranfield University.
[‡]Professor., DoIS, Cranfield University.
[§]Professor., DoIS, Cranfield University.

Submitted to:
Formal Methods for Aerospace 2009

using simulation and testing. They only ensure that the system works for the inputs they are tested for. Testing will demonstrate the presence of bugs, but will not demonstrate the absence of bugs. Even if the system passes all the testing, we can't claim that the system is completely free from errors, as no amount of testing is exhaustive enough.

The mission planning software of multiple UAV systems involves concurrency as it deals with multiple UAVs. It is also reactive, as it constantly interacts with the environment in which the UAVs operate. It is impossible to completely verify such a software system using traditional testing. In addition, concurrency bugs are one of the most difficult ones to test in a traditional way. Moreover, autonomous systems operate in harsh and unpredictable environments and it is difficult to predict before hand the kind of situations that may arise during the mission to carry out testing [8] [6] [3]. Hence, there is a need for formal verification methods, like deductive verification and model checking, which can clarify with high degree of certainty that the system meets its requirements. NASA has been working on developing formal verification techniques for their intelligent autonomous system involving multiple rovers or satellites [9] [19].

Deductive verification is proof-based. It refers to axioms and proof rules to prove the correctness of the system. System description is made in some formal language and leads to a set Γ of appropriate formulas in an appropriate logic. The set Γ constitutes a formal logical inference system for deduction. A system specification is an another formula φ of a chosen logic. The verification consists of finding a proof within the given formal logical system, which would demonstrate that the specification formula φ is inferred from the axioms and inference rules of the formal system Γ, *ie* $\Gamma \models \varphi$. The formal system Γ is assumed to be sound and complete. Deductive verification is well recognized in computer scientists and has significantly influenced the area of software development. However, deductive verification is time-consuming and can be performed only by experts in logic and mathematics.

Model checking, as the name suggests, is model-based. It is an automatic technique for verifying finite state systems. It involves developing a simplified model which captures the essential features of the systems. The specifications which are to be verified on the system are specified, usually in terms of logical statements. Then a model checker, a software tool, systematically examines all the system scenarios to check whether the system satisfies the specifications [5].

In [11] SPIN model checker has been used to verify the safety properties of multi-robot system expressed in Linear Temporal Logic (LTL). In [20] timed automata has been used to model the robots and Upaal model checker has been used to verify the properties of multiple-robot system expressed in Computational Tree Logic (CTL). This paper presents formal modelling of multiple-UAV mission by means of Kripke model and verification of some of the mission properties expressed in CTL. Kripke model offers benefits of using graph theoretic approaches to analyze the system model. SMV model checker is used for verifying the properties, as it is one of the most popular model checkers that supports CTL. In our previous work [23], we reported Kripke model of the behaviour of a single UAV performing a search and verification of its properties expressed in CTL.

The rest of the paper is organized as follows: Section-2 gives an overview of model checking technique. The multiple UAV mission and single UAV behaviour performing the search are discussed in Section-3. Verification of the UAV behaviour using SMV model checker is presented in Section-4.

2 Overview of Model Checking

Model checking is a technique to verify finite state machine abstraction of the system. The system is represented by finite state model M and a temporal logic formula ϕ expressing some desired specification.

A model checker is used to check M against the specification ϕ. The model checkers outputs either true, if M satisfies ϕ *i.e* $M \models \phi$, or a counter example, if it does not. These different steps of model checking are discussed in detail below.

2.1 Model

The first step in model checking is to construct a *formal model* of the system. As model checking can be performed only on finite state systems, the system should be represented as a finite state transition diagram. We are primarily concerned with reactive systems like UAV systems and their behaviour over time. Reactive systems are systems which maintain constant interaction with the environment in which they operate. The family of reactive systems include many classes of programs whose correct and reliable construction is particularly challenging, including concurrent programs, embedded and process control programs, and operating systems. Typical examples of such systems are air traffic control systems, operating systems, and perpetual ongoing processes such as a nuclear reactors.

Reactive systems need to interact with the environment frequently and often do not terminate. Therefore, they can't be adequately modelled by input-output behaviour. Reactive systems can be modelled by capturing the features by means of *state*. A state is an instantaneous description of the system that captures the variables at a particular instant of time. The change from one state to the other as a result of some action determines the *transition* of the system. A *computation* is an infinite sequence of states where each state is obtained from the previous state by some transition.

Kripke structure or *Kripke model* is a type of state transition graph to capture this intuition about the behaviour of reactive systems. Kripke structure consists of a set of states, a set of transitions between the states, and a function that labels each state with a set of properties that are true in that state. Paths in Kripke structure model computations of the system. Although the model of the system is abstract and simple, it should be expressive enough to capture the aspects of temporal behaviour for reasoning about the system. Formal representation of Kripke structure is given below.

A Kripke model is represented by a triplet $M = (S, R, L)$ over a set of atomic propositions AP [15]. A concise representation of a Kripke model whose nodes are states is shown in Figure 1.

1. S is a finite set of states.
2. $R \subseteq S \times S$ is the transition relation.
3. $L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

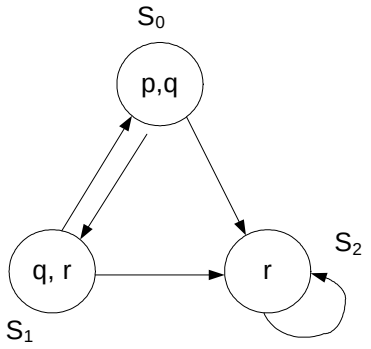

Figure 1: Kripke model representation of a finite state system

2.2 Specification

The properties of the system are usually specified in temporal logic. Temporal logic is a formalism for describing sequences of transitions between states in a reactive system. It is used for specifying and verifying the correctness of digital circuits and computer programs. In classical logic, such as predicate logic, the truth value of a statement is independent of time, whereas in temporal logic the truth value changes dynamically with time. As we are trying to capture the behaviour of multiple UAV group over time, temporal logic suits the purpose of specification language for specifying requirements of the system. There are two fundamental representation types of temporal logic: CTL and LTL. The distinction is how they handle the underlying computation tree.

CTL considers branching of time and allows future paths at any given point of time. The temporal operators quantify over the paths that are possible from a given state. The CTL operators are AX, EX, AG, EG, AU, EU, AF and EF. These operators are pairwise operators. The first of the pair is either A or E. A represents 'along all paths' and E represents 'there exists at least one path'. The second one of the pair is X, F, G, or U meaning 'next state', 'some future state', 'all future states(globally)' and 'until' respectively. CTL has the following syntax given in Backus Naur form:

$$\phi := \bot|\top|p|(\phi)|(\neg\phi)|(\phi\wedge\phi)|(\phi\vee\phi)|\phi\rightarrow\phi|AX\phi|EX\phi|A[\phi\cup\phi]|E[\phi\cup\phi]|AG\phi|EG\phi|AF\phi|EF\phi, \quad (1)$$

where p ranges over atomic formulas.

2.3 Verification

Many automatic model checkers are available, e.g. SMV [16], SPIN[10], KRONOS, HYTECH, NuSMV. SMV model checker has been developed by McMillan in the 90s. It uses SMV language for description of the system model. It accepts temporal specifications expressed either in CTL or LTL. SMV has been used for model checking digital circuits [12], security protocols [14], embedded systems [13] and web applications [17].

3 Mission planning for Multiple UAVs

In recent years there has been a growing interest in employing UAV teams to cooperatively search a given area. The operations of such groups include reconnaissance, surveillance, battle damage assessment, fire monitoring and chemical cloud detection. UAVs are suitable for these operations as they are too dangerous for human pilots. Some of the tasks such as monitoring forest fire propagation and mapping chemical cloud propagation can only be carried out by a group of UAVs and can't be carried out by a single UAV [21], [7], [25], [26].

Hierarchical approach is used extensively for cooperative control because of its simplicity and ease of design [21] [1] [24]. Each layer has different functionalities. This is done to simplify design and to deal with different aspects of the systems in separate layers. Partitioning may result in sub-optimal solution, but as each of these layers have different bandwidths dealing them separately can be justified. The decision making layer operates at very low rate, whereas the path planning layer operates at moderate rate and auto-pilot operates at high rate. The control architecture for multiple UAV mission is shown in Figure 2. The top layer performs as decision making layer taking inputs regarding the state of the UAVs from the middle layer. The middle layer is the path planning and guidance layer. The bottom layer consists of controllers for each UAV.

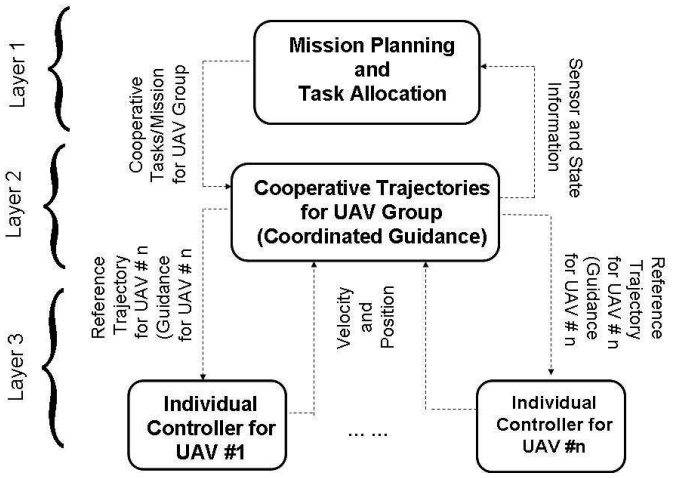

Figure 2: Architecture for multiple-UAV cooperative control

3.1 Cooperative UAV Search

This research is aimed at developing a verifiable multiple UAV mission for cooperatively searching a given area. The mission is to search a given bounded area for static targets and threats using a group of UAVs for the purpose of environmental monitoring. No or little information is available about the area being searched. The UAVs must cooperatively search the environment and mark the positions of targets and threats. Each UAV has a communication link to broadcast the findings and to receive updates from other UAVs of the group. Each UAV has sensors to monitor the environment for the presence of threats and targets. It is assumed that inter-UAV communications are instantaneous, noiseless and have unbounded communication range. It is assumed that each UAV has sufficient processing power for path planning and enough storage to store the global picture of the area being searched. The velocity of the UAVs is 20 m/s and the minimum turn radius is 25 m.

Each UAV stores on board a model of the environment in form of a “search map”. The positions of targets, positions of threats identified by the UAVs in the group and decisions of the other UAVs in the group are stored in this search map. This search map is constantly updated to reflect new information gathered and changes in the state of the UAVs. Sharing information is essential to ensure cooperation for decentralized control approach [2], [26]. As no centralized control is present, sharing of information among the UAVs helps in achieving cooperation. The environment being searched is divided into square cells. Based on the information in search map each UAV will identify an adjacent cell free from threats and other UAVs. The path planning layer generates a path starting from its present position, to the selected neighbouring cell with selected intial and destination headings.

The area to be searched is a square area of 2000X2000 metres. The area is discretized into square cells of 100x100 metres. Discretizing the area into cells helps in reducing the state space and also to visualize the state of location of the UAV. Each cell is identified by coordinates of the centre of cell. The UAVs move through the search area by selecting one of the neighbouring cells. The neighbouring five cells around the cell, in which UAV is flying, are marked as shown in Figure 3 for the purpose of identification. The cell with arrow corresponds to the cell in which the UAV is flying at the time of decision making and the direction of arrow indicates the current heading of the UAV. Cell marked with 1 corresponds to the cell just ahead in the direction of current heading. When cell1 is free, the UAV moves

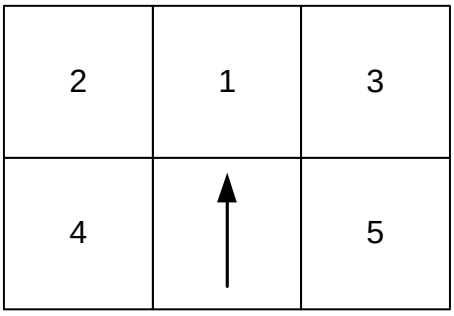

Figure 3: Neighbouring cells marked for decision making

into it. If cell1 contain a threat or it is already chosen by other UAV in the group, the UAV moves into cell3. The order of preference in selecting a neighbouring cell in decreasing order is - cell1, cell3, cell5, cell2 and cell4. When the UAV reaches the north-most cell it moves into cell5. When the UAV reaches the south-most cell it moves into cell4. The UAV flies in a path which connects the centre of present cell and the centre of the chosen neighbouring cell. The initial heading and destination headings of the path are either 90° or 270°. This is done to discretize the heading of the UAV to just two values in order to capture the UAV heading in finite state transition model. The heading is measured with respect to the east. Heading of 90° corresponds to the UAV flying north. The destination heading of the path is same as the initial heading if the UAV selects cell1 or cell2 or cell3. If not, the destination heading is opposite direction to the initial heading. Each UAV in the group repeats this behaviour of selecting a neighbouring free cell.

The decision making layer passes the information of the current and destination cells, current and destination headings to the path planning layer as shown in Figure 4. The path planning layer takes inputs from the decision making layer and generates a flyable path from the current cell to the destination cell with specified starting heading and destination heading. The flight dynamics of the UAV are not taken into account. It is assumed that the flight controller present on the UAV will take inputs from the path planning and guidance layer and follows the generated path.

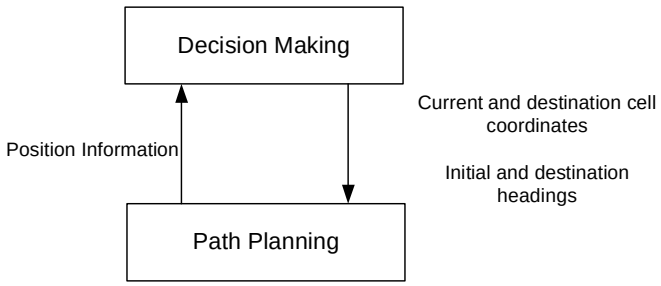

Figure 4: Information exchange between decision making layer and path planning layer

Dubins paths are used to generate path between the way points identified by the decision making layer. The decision making layer passes the inputs to the path planning layer which then produces a Dubins path for the initial and destination poses. Dubins path produces the shortest path between two points by concatenation of circular arcs and their connecting tangents [4] [22]. A straight line is the shortest path between two points. However, for UAVs with constraints on initial heading and final

heading and minimum turning radius, the shortest path is given by concatenation of circular arcs and straight line [4]. Four Dubins curve types *LRL*, *RSR*, *RSL*, *LSR* have been used. *L*, *S*, *R* denote turning left, straight line and turning right respectively. It is assumed that the UAVs fly with a constant altitude. Hence, 2D Dubins paths are used for path planning. 2D Dubins paths for different turning radii and different heading angles are shown in Figure 5.

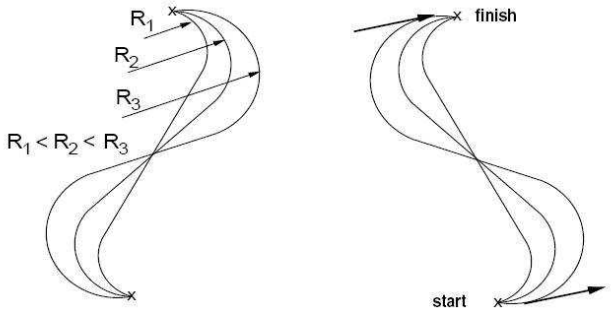

Figure 5: 2D Dubins paths for different turning radii and heading angles

The search strategy discussed above is implemented in `MATLAB`. The trajectories of two UAVs in the group for a flying time of 600 seconds is shown in Figure 6. The red dots indicate the cells with threats.

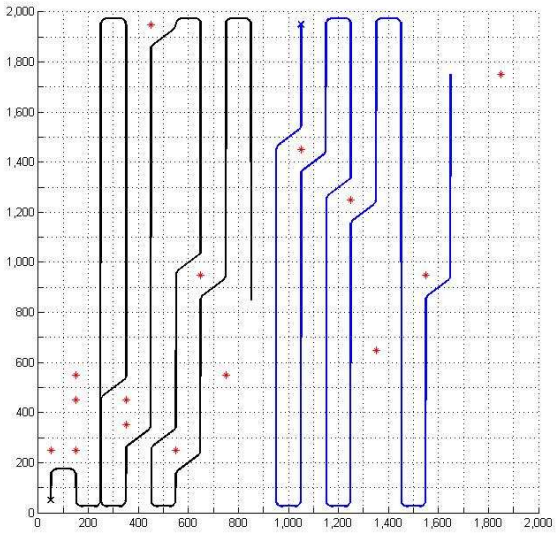

Figure 6: Simulation result of multiple UAV mission

4 Model Checking UAV Misssion

4.1 Kripke Model of UAV Mission

The behaviour of an UAV in the group performing the mission has been captured in Kripke model. CTL has been used to specify the properties because of its expressiveness and ease of modelling in Kripke model. As SMV model checker is a popular model checker for checking CTL properties is has been chosen for our work. SMV language is used for the description of the corresponding Kripke model.

The state of the UAV is captured by the current values of the programs variables. The state of the position of the UAV is captured by means of the coordinates of the centre of the cell. State variables `current_cell` and `destination_cell` capture the present and destination cell coordinates respectively. The initial values for `current_cell`, `initial_heading` are assigned using the keyword `init` as shown in Figure 7. The movement of the UAV from one cell to the next cell is modelled by assigning

```
init(current_cell) := [10 , 10];
init(initial_heading):=90;
```

Figure 7: SMV code showing assignment of initial values

one of the five neighbouring cells to the state variable `cell`. The state variables `initial_heading` and `destination_heading` capture the initial and destination headings of the UAV path connecting the present cell and the destination cell. The destination heading of the path becomes the initial heading for the next path and the destination cell becomes current cell, once the UAV moves to the destination cell. The SMV code modelling these transitions is shown in the Figure 8.

```
next(initial_heading) := destination_heading;
next(current_cell) := destination_cell;
```

Figure 8: SMV code showing transitions

The state variables `north_cell` and `south_cell` model the inputs from the navigation system regarding the present position of the UAV in the search area. The values for `north_cell` and `south_cell` are assigned deterministically as shown in the Figure 9. The state variables like `threat_in_cell1` model

```
north_cell := case
                current_cell[2]=1950:1;
                1:0;
              esac;
south_cell := case
                current_cell[2]=50:1;
                1:0;
              esac;
```

Figure 9: SMV code showing assignment

the environment in which the UAV is operating and model the presence of threats around the UAV. As these variables are purely environmental and the UAV has no control over the location of threats, they

are declared as non-deterministic. The state variables like `other_uav_selected_cell1` model the decisions made by other UAVs in the group. These variables are declared as non-deterministic, as the decisions of the other UAVs are not determined by the state of UAV. As the UAV's decisions are based on the information from the sensors which senses the presence of threats in the immediate neighbourhood, only five cells around the UAV are considered as shown in the Figure 3. For example the presence of threat in the cell11 is modelled by assigning value 1 to the boolean state variable `threat_in_cell1`. The movement of the other UAVs in the group are captured by the boolean state variables. For example the selection of the any UAV in the group to move into cell1 is modelled by assigning 1 to the state variable `other_uav_selected_cell1`.

The destination heading is either 90° or 270° based on the present heading and the destination cell chosen. The coordinates of the destination cell depend on the selection of the neighbouring cell and the coordinates of the current cell. A snippet of SMV code showing the assignment of destination cell coordinates to the state variable `destination_cell` is shown in the Figure 10.

```
destination_cell:=
         case
           cell=cell1  : case
              initial_heading =90 : [current_cell[1] , current_cell[2]+100];
              initial_heading =270: [current_cell[1] , current_cell[2]-100];
                         esac;
           cell=cell2  : ..
                         ..
                         ..
               ..
               ..
         esac;
```

Figure 10: SMV code showing assignment of destination cell coordinates

4.2 Specification and Verification of Properties

CTL is used to express the properties that the UAV mission is expected to satisfy. The property that "*when the UAV is flying with a heading of 90° and not in the north-most cell, then it flies straight when there is no threat ahead and no other UAV has chosen cell1*" is expressed by the formula:

$$AG(\texttt{initial\_heading} = 90 \wedge \neg\texttt{threat\_in\_cell1} \wedge \neg\texttt{other\_uav\_selected\_cell1} \wedge \neg\texttt{north\_cell} \rightarrow \texttt{cell} = \texttt{cell1})$$

Similarly, formula for the case when the UAV heading is 270° is expressed as:

$$AG(\texttt{initial\_heading} = 270 \wedge \neg\texttt{threat\_in\_cell1} \wedge \neg\texttt{other\_uav\_selected\_cell1} \wedge \neg\texttt{south\_cell} \rightarrow \texttt{cell} = \texttt{cell1})$$

The safety property that "*the UAV doesn't enter a cell when either a threat is present or other UAV in the group has already chosen to enter that cell*" is specified by the following formula for the case of

cell1:

$$AG(\texttt{threat\_in\_cell1} \vee \texttt{other\_uav\_selected\_cell1} \rightarrow \neg(\texttt{cell} = \texttt{cell1}))$$

The property that "*the UAV has a initial heading of either 90° or 270° in its path*" is expressed by the following formula

$$AG(\texttt{initial\_heading} = 90 \vee \texttt{initial\_heading} = 270)$$

The Kripke model is verified against the above properties using SMV model checker. Intel 2.2 Ghz processor is used for verificaion and resources used are: user time of 0.046875 sec and system time of 0.015625 sec. SMV produced a output of true for all the above properties. The situation when the UAV is deadlocked and cannot decide where to move next is found by means of violation of the property that "*the UAV chooses one of the five neighbouring cells*". This property can be expressed in CTL as the following formula

$$AG(\neg(\texttt{cell} = \texttt{no\_free\_cell}))$$

The violation of the above property is simulated and the Figure 11 shows a case when the UAV is deadlocked and cannot move any further, as all the neighbouring cells contain threats.

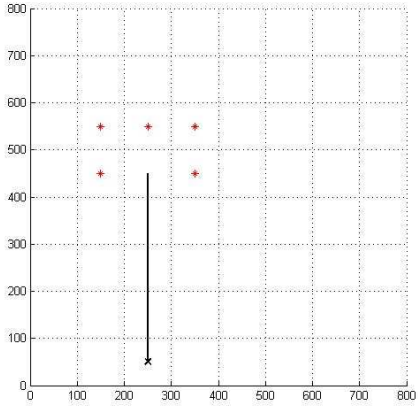

Figure 11: Simulation of a case showing deadlock

5 Conclusion and Future Work

The behaviour of an UAV performing multiple-UAV cooperative search is modelled by Kripke model and some of the properties of the mission are expressed in CTL. SMV model checker is used for verifying the correctness of the model against the temporal specifications. A deadlock has been found and the trace generated by SMV has been simulated. In future, concurrency issues, properties like area coverage, liveness, reachability and fairness have to be verified.

References

[1] Jovan D. Boskovic, Ravi Prakash & Raman K. Mehra (2002): *A multi-layer control architecture for unmanned aerial vehicles*. *Proceedings of the 2002 American Control Conference* , pp. 1825–1830.

[2] Wolfram Burgard, Mark Moors, Cyrill Stachniss & Frank E. Schneider (2005): *Coordinated multi-robot exploration*. *IEEE Transactions on Robotics* 21(3).

[3] Walt Truszkowski Christopher Rouff, Mike Hinchey & James Rash (2005): *Verifying large number of cooperating adaptive agents*. *Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS 2005)* 1, pp. 391–397.

[4] L. E. Dubins (1957): *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents*. *American Journal of Mathematics* 79(3), pp. 497–516.

[5] E.M.Clarke, E.A.Emerson & A.P.Sistla (1986): *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications*. *ACM Transactions on Programming Language and Systems* 8(2), pp. 244–263.

[6] Erann Gat (2004): *Autonomy Software Verification and Validation Might Not Be as Hard as it Seems*. *Proceedings of IEEE Aerospace Conference* , pp. 3123–3128.

[7] Paolo Gaudiano, Eric Bonabeau & Ben Shargel (2005): *Evolving behaviors for a swarm of unmanned air vehicles*. *Proceedings of the 2005 IEEE Swarm Intelligence Symposium* , pp. 317–324.

[8] Michael G.Hinchey, James L.Rash & Christopher A.Rouff (2002): *Verification and validation of autonomous systems*. *Proceedings of 26th Annual NASA Goddard Software Engineering Workshop* .

[9] Ari Jonsson Guillaume Brat (2005): *Challenges in verification and validation of autonmous systems for space exploration*. *Proceedings of IEEE International Joint Conference on Neural Networks* 5, pp. 2909–2914.

[10] G. J. Holzmann (1997): *The Model Checker Spin*. *IEEE Transactions on Software Engineering* 23(5), pp. 279–295.

[11] S. Jayarman, A. Tsourdos, R. Zbikowski & B. White (2006): *Kripke modelling approaches of a multiple robots systems with minimalist communication: a formal approach of choice*. *International Journal of System Sciences* 37(6), pp. 339–349.

[12] Daniela Kotmanova (2008): *Temporal logic in verification of digital circuits*. *Journal of Electrical Engineering* 59(1), pp. 14–21.

[13] Sandeep K.Shukla & Rajesh K.Gupta (2001): *A model checking approach to evaluating system level dynamic power management polocies for embedded systems*. *Proceedings of Sixth IEEE International High-Level Design Validation and Test Workshop* , pp. 53 – 57.

[14] Yuan Lu & Mike Jorda (2004): *Verifying a gigabit ethernet switch using SMV*. *Proceedings of the 41st Annual Conference on Design Automation* , pp. 230– 233.

[15] Edmund M.Clarke, Orna Grumberg & Doron A.Peled (1999): *Model Checking*. *The MIT Press* .

[16] K. L. McMillan (1999): *Cadence. Getting started with SMV*. *Cadence Berkeley Labs* .

[17] Huaikou Miao & Hongwei Zeng (2007): *Model checking-based verification of web applications*. *Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems* , pp. 47–55.

[18] Chang-Su Park, Min-Jea Tahk & Hyochoong Bang (2003): *Multiple aerial vehicles formation using swarm intelligence*. *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit* .

[19] Charles Pecheur (2000): *Verification and validation of autonomy software at NASA*. *NASA/TM 2000-209602* .

[20] M. M. Quottrup, T. Bak & R. Izadi-Zamanabadi (2004): *Multi-robot planning: a timed automata approach*. *Proceedings of the 2004 IEEE International Conference on Robotics and Automation* .

[21] R.W.Beard, T.W.McLain & D.B.Nelson (2007): *Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs*. *Proceedings of the IEEE* 94(7), pp. 1306–1324.

[22] A. M. Shkel & V. Lumelsky (2001): *Classification of the dubins set*. *Robotics and Autonomous Systems* 34(4), pp. 179–274.

[23] Gopinadh Sirigineedi, Antonios Tsourdos, Brian A. White & Rafał Żbikowski (2009): *Towards verifiable approach to mission planning for multiple UAVs*. *Proceedings of AIAA Infotech@Aerospace Conference and AIAA Unmanned .. Unlimited Conference* .

[24] Randal W.Beard, Tinmothy W.McLain, Michael A.Goorich & Erik P.Anderson (2002): *Coordinated target assignment and intercept for unmanned air vehicles*. *IEEE Transactions on Robotics and Automation* 18(6), pp. 911–922.

[25] Y.Jin, A.A.Minai & M.Polycarpou (2003): *Cooperative real-time search and task allocation in UAV teams*. *Proceedings of IEEE Conference on Decision and Control* , pp. 7–12.

[26] Y.Yang, M.Polycarpou & A.A.Minai (2007): *Multi-UAV cooperative search using an opportunistic learning method*. *Transactions of the ASME* 129, pp. 716–728.